

# Ruby on Rails 2.1 What's New?

Ruby on Rails 2.1 の変更点

草稿1 - 未完成

ポルトガル語 原版

**Carlos Brando**  
**Marcos Tapajós**

英語版

**Rafael Barbosa**  
**Caíke Souza**  
**Pedro Pimentel**  
**Abraão Coelho**  
**Ricardo S. Yasuda**

日本語版

**hama**

# まえがき

## はじめに

2004年7月頃に、デビッド・ハイネマイヤー・ハンソン (“*David Heinemeier Hansson*”) が、Basecamp というソフトウェアからスピノフさせた Ruby on Rails フレームワークを一般に公開しました。その約三年後、2007年12月7日に Ruby on Rails はバージョン 2.0 になり、たくさんの重要な変更が加えられました。

そこから6ヶ月の間、世界中から1400人以上の開発者が1600個以上のパッチを提供し、2008年6月1日、Ruby on Rails はバージョン 2.1 になりました。

ハンソンさんによれば、主要な新機能は以下の通りです。

- タイムゾーン
- 値変更の検出 “*dirty tracking*”
- Gem 依存設定 “*gem dependencies*”
- 名前付きスコープ “*named scope*”
- UTC 基準のマイグレーション “*utc-based migrations*”
- キャッシュ機能の改良

新しいバージョンをインストール、またはアップデートをするには、いつもの様に、次のようにコマンドラインで入力します。

```
-----  
gem install rails  
-----
```

## 謝辞

- マルコス・タパヨス (“*Marcos Tapajós*”) へ  
彼が共著者でなければ、この本は存在していませんでした。
- ダニエル・ロペズ (“*Daniel Lopes*”) へ  
美しい表紙を作ってくれました。

- ブラジルの Ruby on Rails コミュニティーの皆さんへ  
ブログにコメントを書いたり、提案をしたりして、間接的・直接的問わず手助けをしてくれた皆さんに感謝します。毎度の様ですが、Rails のもっとも素晴らしい点はそのコミュニティーにあります！これからも制作し、発明し、そして何より共有していきましょう。

## 英語版

英語版は以下のブラジル人達によって原版であるポルトガル語から翻訳されました。

- ラファエル・バルボサ (“Rafael Barbosa”) <http://www.acts-as-newbie.com>  
序章と第1章
- ケイク・ソウザ (“Caíke Souza”) <http://tech-death.blogspot.com>  
第2章
- ペドロ・ピメンテル (“Pedro Pimentel”) <http://www.pedropimentel.com>  
第3章から第8章と、第10章から第13章
- アブバーオ・コエロ (“Abraão Coelho”) <http://abrcoelho.net>  
第9章
- リカルド・Y・ヤスダ (“Ricardo S Yasuda”) <http://blog.shadowmaru.org>  
第14章

## 英語版 謝辞

- ジョルディ・バンスター (“Jordi Bunster”) <http://bunster.org>

## 日本語訳

この日本語訳は hama によって、英語版を元にして翻訳されました。

### ★ 訳者より

この日本語版は、ポルトガル語である原版をブラジル人達の手によって訳された英語版を元に再度日本語に訳したものですので、原版とはかなり違う表現になっている可能性があります。しかし、説明の根幹自体は保たれていると思います。

また、私なりに言い回しを変えてあり、一対一の完全な翻訳とは言えない部分が多くあります。翻訳と呼ぶのにはふさわしくないかもしれませんが、やはり、説明の根幹自体は失われない様に気をつけましたので、本の目的である、変更点を伝えること自体は達成できていると思います。

誤訳や問題点などが見つかりましたら、お手数ですが hama までご連絡ください。以下のメールアドレスが私の連絡先です。

[hama@yoidore.org](mailto:hama@yoidore.org)

未熟者の翻訳なので何かと読みづらいかもしれませんが、Ruby on Rails 2.1 の変更点を把握するには役に立つと思います。どうぞよろしくお願いします。

2008年6月13日 hama

★ 注意点

日本語版では、英語版の第1章を「はじめに」に置き換えています。それにあわせて、英語訳の第2章 `ActiveRecord` が日本語訳の第1章にあたり、英語版の第15章 `CHANGELOG` が日本語版の第14章にあたります。

# 目次

## 第1章 11

### **ActiveRecord 11**

- ★ `sum` メソッド 11
  - 式を受け取る `sum` 11
  - `sum` のデフォルト戻り値 11
- ★ `has_one` メソッド 11
  - `has_one :through` に対応 11
  - `has_one` の `:source_type` オプション 12
- ★ `named_scope` メソッド 13
- ★ インクリメントとデクリメント 14
- ★ `find` メソッド 14
- ★ `find` の `:last` オプション 15
- ★ `all`, `first`, `last` メソッド 15
- ★ 積極的読み込み (“eager loading”) 16
- ★ `belongs_to` メソッド 17
- ★ 多形態 URL 17
- ★ 読み取り専用の関連付け 18
- ★ `add_timestamps` と `remove_timestamps` メソッド 18
- ★ 計算メソッド 18
- ★ ブロックを利用した `create` メソッドの呼び出し 19
- ★ `change_table` メソッド 20
- ★ 値変更の検出 (“dirty objects”) 20
- ★ 部分的書き込み 22
- ★ MySQL における Integer サイズの自動検出 23
- ★ `has_one` と `belongs_to` の `:select` オプション 24
- ★ 単一テーブル継承の完全クラス名保存 24

- ★ `table_exists?` メソッド 25
- ★ タイムスタンプ基準のマイグレーション 25

## 第2章 27

### ActiveSupport 27

- ★ ActiveSupport::CoreExtensions::Date::Calculations モジュール 27
  - Time#end\_of\_day メソッド 27
  - Time#end\_of\_week メソッド 27
  - Time#in\_time\_zone メソッド 27
  - Time#days\_in\_month メソッド 27
- ★ DateTime#to\_f メソッド 28
- ★ Date.current メソッド 28
- ★ fragment\_exist? メソッド 28
- ★ UTC なのか GMT なのか 28
- ★ json\_escape メソッド 29
- ★ mem\_cache\_store のオプション 29
- ★ Time.current メソッド 30
- ★ 余計なスペースを削除する squish メソッド 30

## 第3章 31

### ActiveResource 31

- ★ メールアドレスをユーザー名にできる 31
- ★ clone メソッド 31
- ★ タイムアウト設定 32

## 第4章 33

### ActionPack 33

- ★ タイムゾーン 33
  - タイムゾーンの設定 33

- formatted\_offset メソッド 33
- with\_env\_tz メソッド 34
- Time.zone\_reset! メソッド 34
- Time#in\_current\_time\_zone メソッド 34
- Time#change\_time\_zone\_to\_current メソッド 34
- TimeZone#now メソッド 35
- compare\_with\_coercion メソッド 35
- TimeWithZone#between? メソッド 35
- TimeZone#parse メソッド 35
- TimeZone#at 36
- その他のタイムゾーン関連メソッド 36
- Ruby 1.9 に向けて 36
- ★ 自動リンク auto\_link メソッド 37
- ★ form\_for メソッドのラベル 38
- ★ 部分テンプレート呼び出し 39
- ★ ATOM フィードの新しいネームスペース 40
- ★ キャッシュ 41
- ★ String.titleize メソッドのバグ 43
- ★ action\_name メソッド 43
- ★ 条件付き caches\_action メソッド 43
- ★ 条件付き caches\_page メソッド 44
- ★ テストできる様になったフラッシュメッセージ 44
- ★ ビュー外でも利用できる様になったヘルパー 44
- ★ JSON を利用した POST リクエスト 45
- ★ パス名 45
- ★ ルートファイルの場所指定 45
- ★ session メソッドの :on と :off 引数 46
- ★ 簡単になったヘルパーのテスト 46

## **ActionController 48**

- ★ ActionController::Routing モジュール 48
  - map.root メソッド 48
  - Rails によるルートの認識と構築 48
  - assert\_routing テストメソッド 48
  - map.resources メソッド 49
- ★ ActionController::Caching::Sweeping モジュール 49

## **第6章 51**

### **ActionView 51**

- ★ ActionView::Helpers::FormHelper モジュール 51
  - field\_for と form\_for メソッドの :index オプション 51
- ★ ActionView::Helpers::DateHelper モジュール 51
  - date\_helper メソッド 52
- ★ ActionView::Helpers::AssetTagHelper 52
  - register\_javascript\_expansion メソッド 52
  - register\_stylesheet\_expansion メソッド 52
- ★ ActionView::Helpers::FormTagHelper 53
  - submit\_tag メソッド 53
- ★ ActionView::Helpers::NumberHelper 53
  - number\_to\_currency メソッド 53
- ★ ActionView::Helpers::TextHelper 54
  - excerpt メソッド 54
  - simple\_format メソッド 54

## **第7章 55**

### **Railties 55**

- ★ config.gem メソッド 55
- ★ プラグインの config.gem メソッド 56
- ★ gems:build タスク 56

- ★ サーバー起動時のメッセージ 56
- ★ Rails.public\_path 変数 57
- ★ Rails.logger、Rails.root、Rails.env と Rails.cache 57
- ★ Rails.version 変数 57
- ★ プラグインの情報を取得する 58

## 第8章 59

### Rake タスク、プラグイン、スクリプト 59

- ★ Rake タスク 59
  - rails:update タスク 59
  - 127.0.0.1 のデータベース 59
  - 指定したバージョンの Rails を固める 59
- ★ タイムゾーン 59
  - time:zones:all タスク 59
  - time:zones:us タスク 59
  - time:zones:local タスク 59
- ★ スクリプト 60
  - plugin スクリプト 60
  - dbconsole スクリプト 60
- ★ プラグイン 60
  - gem のままプラグインをインストール 60
  - 標準パス外プラグインのジェネレーター 60

## 第9章 61

### prototype.js と script.aculo.us 61

## 第10章 62

### Ruby 1.9 に向けて 62

- ★ 詳細 62

- ★ DateTime クラスの新しいメソッド 62

## 第11章 63

### デバッグ 63

- ★ ruby-debug の標準化 63

## 第12章 64

### バグの修正 64

- ★ PostgreSQL 利用時の add\_column メソッド 64
- ★ MIME タイプ 65
- ★ change\_column メソッド 65

## 第13章 66

### 追加情報 66

- ★ クロスサイトスクリプティング対策 66
- ★ method\_missing を応用したメソッドの注意点 67
- ★ PostgreSQL 7.4 以上に対応 68

## 第14章 69

### 変更履歴 (“CHANGELOG”) 69

- ★ ActionMailer 69
- ★ ActionPack 69

# 第1章

## ActiveRecord

Ruby on Rails の ActiveRecord は、アプリケーション・データベース間の相互運用性を提供するオブジェクト関係マッピング層で、抽象化されたデータを提供します。

ウィキペディアより

### ★ sum メソッド

#### 式を受け取る sum

新しいバージョンでは、ActiveRecord のメソッドに計算式を与えられる様になりました。例えば、sum メソッドでは以下の様なことができるようになりました。

```
-----  
Person.sum("2 * age")  
-----
```

#### sum のデフォルト戻り値

以前のバージョンでは、ActiveRecord の sum メソッドに条件を与えた時、その条件にどのレコードもマッチしなかった場合、nil が返されていました。Rails 2.1 では、その値が以下のように、0 に変更されています。

```
-----  
Account.sum(:balance, :conditions => '1 = 2') #=>  
-----
```

### ★ has\_one メソッド

#### has\_one :through に対応

has\_one も、has\_many の様に :through オプションに対応しました。機能的には has\_many :through とほとんど同じですが、一件の ActiveRecord オブジェクトにのみ関係を持ちます。

```
-----  
class Magazine < ActiveRecord::Base  
  has_many :subscriptions  
end  
-----
```

```

class Subscription < ActiveRecord::Base
  belongs_to :magazine
  belongs_to :user
end

class User < ActiveRecord::Base
  has_many :subscriptions
  has_one :magazine, :through => :subscriptions,
    :conditions => ['subscriptions.active = ?', true ]
end

```

## has\_one の :source\_type オプション

上記の has\_one :through メソッドは、:source\_type も受け取る様になりました。例を挙げて説明しましょう。まず、以下の様なクラスがあるとします。

```

class Client < ActiveRecord::Base
  has_many :contact_cards
  has_many :contacts, :through => :contact_cards
end

```

この Client クラスは、多形態関係 “*polymorphic relationship*” を利用した ContactCard クラスを通じて、複数の contacts を持っています。次に ContactCard クラスとなる二つのクラスを追加します。

```

class Person < ActiveRecord::Base
  has_many :contact_cards, :as => :contact
end

class Business < ActiveRecord::Base
  has_many :contact_cards, :as => :contact
end

```

Person と Business クラスは、ContactCard クラスを通じて Client クラスと関係を持っています。言い換えれば、Person と Business という二つの contacts がということになります。しかし、これはうまくいきません。contacts を参照しようとすると以下の様になります。

```

>> Client.find(:first).contacts
# ArgumentError: /.../active_support/core_ext/hash/keys.rb:48:
# in 'assert_valid_keys': Unknown key(s): polymorphic

```

こうならない様にする為には、`:source_type` を指定します。Client クラスに `:source_type` を追加します。

```
-----  
class Client < ActiveRecord::Base  
  has_many :people_contacts,  
           :through => :contact_cards,  
           :source => :contacts,  
           :source_type => :person  
  has_many :business_contacts,  
           :through => :contact_cards,  
           :source => :contacts,  
           :source_type => :business  
end  
-----
```

これで、それぞれの `:source_type` に対応した `contacts` を別々に参照できるようになります。

```
-----  
Client.find(:first).people_contacts  
Client.find(:first).business_contacts  
-----
```

#### ★ `named_scope` メソッド

Rails に追加された新機能 `named_scope` を理解する為に、以下の例を見てみましょう。

```
-----  
class Article < ActiveRecord::Base  
  named_scope :published, :conditions => { :published => true }  
  named_scope :containing_the_letter_a,  
              :conditions => "body LIKE '%a%'"  
end  
  
Article.published.paginate(:page => 1)  
Article.published.containing_the_letter_a.count  
Article.containing_the_letter_a.find(:first)  
Article.containing_the_letter_a.find(:all, :conditions => {...})  
-----
```

ここでは `published` が `true` に設定されているすべての `Article` や、`body` に `a` を含むすべての `Article` を取得するメソッドを作るかわりに、`named_scope` を使っています。`named_scope` はこれだけではありません。次の様にも使うことができます。

```
-----  
named_scope :written_before, lambda { |time|  
  { :conditions => ['written_on < ?', time] }  
}
```

```
}

named_scope :anonymous_extension do
  def one; 1 end
end

named_scope :named_extension, :extend => NamedExtension
named_scope :multiple_extensions, :extend => [MultipleExtensionTwo,
MultipleExtensionOne]
```

---

#### ★ インクリメントとデクリメント

ActiveRecord のメソッド `increment` と `decrement`、そしてそれぞれの破壊的なメソッド `increment!` と `decrement!` が、オプションを受け取るようになりました。以前は、指定した項目に対する 1 の足し引きしかできませんでしたが、Rails 2.1 では以下の様に足し引きする数値を変更できるようになりました。

```
player1.increment!(:points, 5)
player2.decrement!(:points, 2)
```

---

ここでは、`player1` に5点を足し、`player2` から2点を引いています。このオプションは任意なので、このオプションを指定していない既存のコードには影響はありません。

#### ★ find メソッド

ActiveRecord の `find` メソッドに引数としてオブジェクトを渡せるようになりました。以下の例を見てください。

```
class Account < ActiveRecord::Base
  composed_of :balance, :class_name => "Money", :mapping => %w(balance
amount)
```

---

この時、`Account` クラスの `find` メソッドに対して以下の様に `Money` クラスのインスタンスを渡せるようになります。

```
amount = 500
currency = "JPY"
Account.find(:all, :conditions => { :balance => Money.new(amount,
currency) })
```

---

## ★ find の :last オプション

今までの ActiveRecord の find メソッドでは、レコードを参照する方法が :first と :all、そしてレコードの id で直接指定する三つの方法しかありませんでした。Rails 2.1 では、四つ目となる :last が追加されます。以下がその例です。

```
-----  
Person.find(:last)  
Person.find(:last, :conditions => [ "user_name = ?", user_name ] )  
Person.find(:last, :order => "created_on DESC", :offset => 5)  
-----
```

以下のテストケースが、この :last の挙動を理解する助けになります。

```
-----  
def test_find_last  
  last = Developer.find :last  
  assert_equal last, Developer.find(:first, :order => 'id desc')  
end  
-----
```

## ★ all, first, last メソッド

静的メソッドの all と first、そして last メソッドが、同じように静的メソッドである find を通じて対応するレコードを返すエイリアスとして定義されています。(訳注)

```
-----  
Topic.all # == Topic.find(:all)  
Topic.first # == Topic.find(:first)  
Topic.last # == Topic.find(:last)  
-----
```

また、これらは named\_scope に対しても利用可能です。Post という ActiveRecord のクラスがあり、recent という named\_scope を持っているとした時、以下の様な参照は可能です。

```
-----  
Post.comments.recent.last  
-----
```

## 訳注

all、first、last メソッドがそれぞれ独立して説明され、named\_scope でも使えるとさらに独立して説明されていますが、関連した内容なので一つにまとめました。

★ 積極的読み込み (“eager loading”)

この新機能の説明として、以下の例を見てみましょう。

```
-----  
Author.find(:all, :include => [:posts, :comments])  
-----
```

この呼び出しでは、慣習通りに設定された `author_id` を通じて関連付けられた `posts` と `comments` を含むすべてのレコードが `authors` テーブルから読み込まれます。この際、今までは以下の様な SQL クエリーが発行されていました。

```
-----  
SELECT  
  authors."id" AS t0_r0,  
  authors."created_at" AS t0_r1,  
  authors."updated_at" AS t0_r2,  
  posts."id" AS t1_r0,  
  posts."author_id" AS t1_r1,  
  posts."created_at" AS t1_r2,  
  posts."updated_at" AS t1_r3,  
  comments."id" AS t2_r0,  
  comments."author_id" AS t2_r1,  
  comments."created_at" AS t2_r2,  
  comments."updated_at" AS t2_r3  
FROM  
  authors  
  LEFT OUTER JOIN posts ON posts.author_id = authors.id  
  LEFT OUTER JOIN comments ON comments.author_id = authors.id  
-----
```

一件の、outer join を利用して `authors` と `posts`、そして `comments` をまたいだ、長い SQL クエリーが発行されます。ちなみにこれを直積結合と呼びます。

しかし、この様なクエリーは、パフォーマンスが良くないことが多いため、Rails 2.1 では変更が加えられました。同じ呼び出しを `Author` クラスに行った時、別の方法で三つのテーブルからレコードを読み込みます。新しい Rails は、すべてのテーブルを利用した一件のクエリーのかわりに、各テーブルに対して一件ずつ、合計三件の短いクエリーを発行します。`Author` クラスに先ほどと同じ呼び出しを行った際、以下のようなクエリーがログに残ります。

```
-----  
SELECT * FROM "authors"  
SELECT posts.* FROM "posts" WHERE (posts.author_id IN (1))  
SELECT comments.* FROM "comments" WHERE (comments.author_id IN (1))  
-----
```

多くの場合、この様な短い三件のクエリーの方が、先ほどの様な一件の長く複雑なクエリーより早く処理されます。

#### ★ belongs\_to メソッド

belongs\_to メソッドによる関連付けにおいて、:dependent => :destroy と :delete が指定できる様になりました。以下が利用例です。

```
-----  
belongs_to :author_address  
belongs_to :author_address, :dependent => :destroy  
belongs_to :author_address_extra, :dependent => :delete,  
  :class_name => "AuthorAddress"  
-----
```

#### ★ 多形態 URL

多形態 URL (“*polymorphic URL*”) へのヘルパーメソッドを利用して、より美しく名前付きルート “renamed route” を生成できる様になりました。このメソッドは RESTful なリソースの URL を生成する際に便利で、関連付けられたクラスを指定せずに済みます。利用方法は以下の通りで、簡単です。（コメント部分は Rails 2.1 以前で同じことをする際のコードです。）

```
-----  
record = Article.find(:first)  
polymorphic_url(record) # -> article_url(record)  
record = Comment.find(:first)  
polymorphic_url(record) # -> comment_url(record)  
  
# 新しいレコードであることも認識します。  
  
record = Comment.new  
polymorphic_url(record) # -> comments_url()  
-----
```

この様に、polymorphic\_url メソッドは渡されたオブジェクトのクラスを自動的に認識して適切なルートを生成します。入れ子リソース “nested resource” や、ネームスペースにも対応し、以下の様な利用もできます。

```
-----  
polymorphic_url([:admin, @article, @comment])  
  
# -> この呼び出し結果は、以下の呼び出し結果と同じです。  
admin_article_comment_url(@article, @comment)  
-----
```

また、`new` や `edit` もしくは `formatted` といった接頭語を以下の様に加えて、対応したルートを生成させることもできます。

```
-----  
edit_polymorphic_path(@post) # => /posts/1/edit  
formatted_polymorphic_path(@post) # => /posts/1.pdf  
-----
```

#### ★ 読み取り専用の関連付け

モデルの関連付けにも新しい機能が追加されました。関連付けられたモデルの値に変更が加えられない様にする場合、次の様に `:readonly` オプションを指定できるようになりました。

```
-----  
has_many :reports, :readonly => true  
has_one :boss, :readonly => :true  
belongs_to :projects, :readonly => true  
has_and_belongs_to_many :categories, :readonly => true  
-----
```

この様に設定することで、あるモデルに関連付けられたモデルがそのモデルからの参照によって変更できなくなり、万が一変更が加えられた場合に `ActiveRecord::ReadOnlyRecord` 例外を投げる様になります。

#### ★ `add_timestamps` と `remove_timestamps` メソッド

`ActiveRecord` のマイグレーションに `add_timestamps` と `remove_timestamps` が追加されました。以下の様にマイグレーション内で利用すると、名前の通りタイムスタンプ項目の追加と削除を行います。

```
-----  
def self.up  
  add_timestamps :feeds  
  add_timestamps :urls  
end  
  
def self.down  
  remove_timestamps :urls  
  remove_timestamps :feeds  
end  
-----
```

#### ★ 計算メソッド

`ActiveRecord::Calculations` で提供されるメソッドで、テーブル名を指定できる様に若干の変更が加えられました。同じ項目名を持った複数のテーブルに関連付けている際に、テーブ

ル名を明示的に指定できるようになります。結果、それらのメソッドは以下の様な二つの呼び出し方法が利用可能です。

```
-----  
authors.categories.maximum(:id)  
authors.categories.maximum("categories.id")  
-----
```

#### ★ ブロックを利用した create メソッドの呼び出し

ActiveRecord::Base.new メソッドは以前からブロックを受け取っていましたが、Rails 2.1 では、同じように create メソッドも以下の様にブロックを受け取る様になりました。

```
-----  
# ブロックで各項目の値を指定して新しいオブジェクトを作成する。  
User.create(:first_name => 'Taro') do |u|  
  u.is_admin = false  
end  
-----
```

同じメソッドを利用して複数のオブジェクトをまとめて作成することもできます。

```
-----  
# ブロックで各項目の値を指定して新しいオブジェクトを作成する。  
# ブロック部分は渡された配列内のハッシュを使って作られたそれぞれのオブジェクトの値を上書きする。  
  
User.create([{:first_name => 'Taro'}, {:first_name => 'Hanako'}]) do |u|  
  u.is_admin = false  
end  
-----
```

関連付けにも対応しています。

```
-----  
author.posts.create!(:title => "最新版") { |p| p.body = "さらに便利!" }  
# {} を利用しても do を利用してももちろん同じ結果です。  
author.posts.create!(:title => "最新版") do |p|  
  p.body = "さらに便利!"  
end  
-----
```

## ★ change\_table メソッド

マイグレーションによるテーブルの作成は、 Rails 2.0 でとても美しくなりました。しかし、テーブルの変更はまったくもってそうではありませんでした。幸いにも Rails 2.1 では、change\_table メソッドが追加され、テーブルの変更が次の例の様に美しくなりました。

```
-----  
change_table :videos do |t|  
  t.timestamps # タイムスタンプ (created_at, updated_at) の追加  
  t.belongs_to :goat # goat_id の追加  
  t.string :name, :email, :limit => 20 # name, email の追加  
  t.remove :name, :email # 上の name, email の削除  
end  
-----
```

このメソッドは、仲間の create\_table メソッドと同じ様に呼び出すことが可能で、ただ一つ違うのは、既存のテーブルへの項目やインデックスの追加と削除のみを扱うという点です。以下がブロックに渡されるオブジェクトで提供されるメソッドです。

```
-----  
change_table :table do |t|  
  t.column # 例: name という文字列項目を追加するには t.column :name, :string  
  t.index # 新しいインデックスを追加する。  
  t.timestamps # タイムスタンプを追加する。  
  t.change # 項目の詳細を変更する。例: name という項目を80文字までの文字列に変更するには t.change(:name, :string, :limit => 80)  
  t.change_default # 項目の初期値を変更する。  
  t.rename # 項目の名前を変更する。  
  t.references; t.belongs_to; t.string; t.text; t.integer; t.float  
  t.decimal; t.datetime; t.timestamp; t.time; t.date; t.binary  
  t.boolean; t.remove; t.remove_references; t.remove_belongs_to  
  t.remove_index; t.remove_timestamps  
end  
-----
```

## ★ 値変更の検出 (“dirty objects”)

新しい Rails では、ActiveRecord オブジェクトの値が変更された(訳注)かどうかを検出することができ、変更された場合は、変更前の値と変更後の値をそれぞれ参照することができます。以下の例を見てみましょう。

```
-----  
article = Article.find(:first)  
article.changed? # => false  
article.title #=> "Title"  
article.title = "New Title"  
article.title_changed? # => true  
article.title_was # => "Title" # 変更前の title を参照する。  
  
article_title_change # => ["Title", "New Title"] # 変更前と変更後を参照する。  
-----
```

この様に、とても簡単に変更を確認することができます。そのオブジェクトの変更すべてを参照することも、以下の二つの方法で可能です。

```
-----  
article.changed # => ['title'] # 変更された項目名を参照する。  
# ハッシュで変更された項目名と変更内容を参照する。  
article.changes # => { 'title' => ["Title", "New Title"] }  
-----
```

オブジェクトが save メソッドなどを通じてデータベースに保存されると、状態が更新されます。

```
-----  
article.changed? # => true  
article.save # => true  
article.changed? # => false  
-----
```

もし、attr= メソッド以外、上の例で言うところの title= メソッド以外で値を変更する場合は、変更前にあらかじめ attr\_name\_will\_change! (attr は項目名) を、次の例の様に呼び出しておかなければ変更は記録されません。

```
-----  
article = Article.find(:first)  
article.title_will_change! # 明示的に変更されることを宣言  
article.title.upcase!  
article.title_change # => ["title", "TITLE"]  
-----
```

## 訳注

変更が加えられたがデータベースにまだ保存されておらず、整合性が取れていないオブジェクトのことを汚れたオブジェクト ("dirty objects") と呼んでいます。

## ★ 部分的書き込み

値変更の検出が実装されたことによって、とても興味深い他の機能の実装が可能になりました。今までの Rails で既存のオブジェクトの `save` メソッドを呼び出すと、変更されていない項目も含め、オブジェクトの全項目をデータベースに保存していました。しかし、バージョン 2.1 では値変更の検出のおかげで、データベース保存時に、変更された値だけを更新する様に設定することが可能です。Rails 2.1 において、若干の項目に変更が加えられ、保存する際に発行される次の SQL クエリーを見てみましょう。

```
-----  
article = Article.find(:first)  
article.title # => "Title"  
article.subject # => "Edge Rails"  
  
# タイトルを変更します。  
article.title = "New Title"  
article.save # 次の様な SQL クエリーを発行します。  
# => "UPDATE articles SET title = 'New Title' WHERE id = 1"  
-----
```

この様に、アプリケーション中で変更された項目だけがデータベース上で変更されます。もし、どの項目も変更されていないければ、ActiveRecord は SQL クエリー自体を発行しません。この機能を有効にするには、モデルの `partial_updates` プロパティを変更する必要があります。

```
-----  
MyClass.partial_updates = true # 部分的書き込みを有効にする。  
-----
```

もし、すべての ActiveRecord クラスでこの機能を有効にしたければ、`config/initializers/new_rails_defaults.rb` 内で以下の様に設定します。

```
-----  
# すべてのモデルで部分的書き込みを有効にする。  
ActiveRecord::Base.partial_updates = true  
-----
```

部分的書き込みを有効にした時に忘れてはならないのが、もし `attr=` メソッド以外で項目を更新する場合は、事前に `attr_will_change!` を呼び出しておくことです。

```
-----  
# **attr=** メソッドを使う場合は問題ありませんが...  
person.name = 'bobby'  
-----
```

```
person.name_change # => ['bob', 'bobby']
```

# **\*\*attr\*\*** メソッドを使わない場合は、明示的に変更が加えられることを宣言してください。

```
person.name_will_change!  
person.name << 'by'  
person.name_change # => ['bob', 'bobby']
```

なぜ忘れてはならないかというと、部分的書き込みを有効にした場合、ActiveRecord が変更を検出できなければ、仕様上 save メソッドを呼び出してもデータベースに変更が反映されなくなってしまうからです。

#### ★ MySQL における Integer サイズの自動検出

ActiveRecord の MySQL アダプターは賢くなりました。マイグレーションにおいて、Integer 項目を設定すると自動的に :limit オプションを参照し、その項目が MySQL データベース上で、smallint、int と bigint のうちどれであるべきかを以下の条件で判断し、設定します。

```
case limit  
when 0..3  
  "smallint(#{limit})"  
when 4..8  
  "int(#{limit})"  
when 9..20  
  "bigint(#{limit})"  
else  
  'int(11)'  
end
```

マイグレーションファイルで言うと以下の様になります。

```
create_table :table_name, :force => true do |t|  
  # 0~3: smallint  
  t.integer :coluna1, :limit => 2 # smallint(2)  
  # 0~8: int  
  t.integer :coluna2, :limit => 6 # int(6)  
  # 9~20: bigint  
  t.integer :coluna3, :limit => 15 # bigint(15)  
  # :limit が定義されていない場合は int(11)
```

```
t.integer :coluna4 # init(11)
end
```

-----

実は PostgreSQL アダプターには以前からこの機能が備わっていて、MySQL アダプターが Rails 2.1 で追いついたのです。

★ has\_one と belongs\_to の :select オプション

説明不要の has\_one メソッドと belongs\_to メソッドに新しいオプション :select が追加され、読み込みたい項目を指定できるようになりました。デフォルトの値は “SELECT FROM table” である様に “” (空の String) で、今まで通り、すべての項目を読み込みます。忘れないでおきたいのが、もし指定する場合は、プライマリーキーと外部キーを含めておくことです。そうでないと、エラーが出ます。

関連することですが、Rails 2.1 では、belongs\_to の :order オプションが無くなりました。もともと使い道のないオプションだったので、さほど気にすることはありません。

★ 単一テーブル継承の完全クラス名保存

今までの Rails では、ネームスペース内にあるモデルで単一テーブル継承 (STI: “single table inheritance”) を利用すると、ActiveRecord はクラス名だけを type 項目に保存し、ネームスペース名を省略していました。例えば、以下の様な問題が発生していました。

```
-----
class CollectionItem < ActiveRecord::Base; end
class ComicCollection::Item < CollectionItem; end

item = ComicCollection.Item.new
item.type # => 'Item'

item2 = CollectionItem.find(item.id)
# Item クラスは存在しないのでエラーが発生する。
-----
```

これを回避する為に追加された新しい機能で、ActiveRecord がネームスペース名も合わせた完全クラス名を保存する様になりました。設定はデフォルトで有効になっており、何らかの理由でこれを無効にする場合は environment.rb 内で以下の様に設定します。(訳注)

```
-----
ActiveRecord::Base.sore_full_sti_class = false
-----
```

## 訳注

デフォルトで有効になっているにも関わらず、有効にする方法が説明されていたので、逆の無効にする方法を説明しています。

### ★ table\_exists? メソッド

AbstractAdapter クラスに、データベースのテーブルが存在するかどうかを確認する table\_exists? メソッドが追加されました。使い方は非常に簡単です。

```
-----  
>> ActiveRecord::Base.connection.table_exists?("users")  
=> true  
-----
```

### ★ タイムスタンプ基準のマイグレーション

Rails について学んでいたたり、一人で開発している場合は、マイグレーションが非常に便利なものに見えるかもしれませんが、しかし、チームの一部になり複数人で開発していて、皆がマイグレーションを書いている場合では、実際にやればわかると思いますが、マイグレーションは使い物になりません。Rails 2.1 でこの問題は解決されました。

マイグレーションは、作成時に割り振られる番号で管理されていました。しかし、二人の開発者同時にマイグレーションを作成したらどうなるでしょうか。複数のメンバーがマイグレーションを作り、その日の終わり頃にコミットをしたらどうなるでしょうか。結果、同じ番号が割り振られた複数のマイグレーションが存在してしまい、競合が生まれてしまいます。

この重大な問題を解決するために、Rails のコア開発チームはマイグレーションの管理方法に変更を加えました。Rails 2.1 では、順番に割り振られた番号ではなく、UTC 基準のタイムスタンプ (YYYYMMDDHHMMSS) で管理する様になっています。さらに、新しく追加された schema\_migrations テーブルに、既に行われたマイグレーションを記録する様になりました。これで、適用待ちのマイグレーションがある中で、他の誰かが新しいマイグレーションを作成したとしても、db:migrate タスクがどのマイグレーションを実行すべきか適切に判断できる様になります。逆も同じように、マイグレーション取り消しを行う際に、適用していないマイグレーションまで取り消したりすることが無くなります。

これらの変更によって、マイグレーションの競合問題は解決されました。何かしらの理由でこの機能を無効にしたい場合は、environment.rb に以下の設定を追加することで無効にできます。

---

```
config.active_record.timestamped_migrations = false
```

---

また、マイグレーションを適用したり取り消したりする為の、以下の様な新しい rake タスクが追加されました。

---

```
rake db:migrate:up  
rake db:migrate:down
```

---

# 第2章

## ActiveSupport

ActiveSupport は、Ruby on Rails で開発されたアプリケーションに役立つであろう便利なクラスと標準ライブラリの拡張を集めたパッケージです。

ウィキペディアより

★ ActiveSupport::CoreExtensions::Date::Calculations モジュール

### Time#end\_of\_day メソッド

その日の午後 11:59:59 を返すようになりました。

### Time#end\_of\_week メソッド

その週の終わり日曜日の午後 11:59:59 を返すようになりました。

### Time#in\_time\_zone メソッド

このメソッドは、OS のタイムゾーンを利用するかわりに Time.zone を利用する点以外、Time#localtime に似ています。TimeZone か String オブジェクトを渡すことができます。

```
-----  
Time.zone = 'Hawaii'  
Time.utc(2000).in_time_zone  
# => Fri, 31 Dec 1999 14:00:00 HST -10:00
```

```
Time.utc(2000).in_time_zone('Alaska')  
# => Fri, 31 Dec 1999 15:00:00 AKST -09:00  
-----
```

### Time#days\_in\_month メソッド

days\_in\_month メソッドにあった、年を指定していない場合でかつ2月の場合に、返される数値が間違っているバグが修正されました。この問題は、年が指定されていない場合、その年をデフォルトの数値として利用することで修正されました。

今年が閏年であると仮定すると以下のようになります。

---

```
Loading development environment (Rails 2.0.2)
>> Time.days_in_month(2)
=> 28
```

```
Loading development environment (Rails 2.1.0)
>> Time.days_in_month(2)
=> 29
```

---

#### ★ DateTime#to\_f メソッド

DateTime クラスに、その時点の UNIX epoch (1970/1/1 00:00) からの秒数を返す、新しい to\_f メソッドが追加されました。

#### ★ Date.current メソッド

Date クラスに、Rails 2.1 からは Date.today のかわりとして使われるべきである、新しいメソッド current が追加されました。このメソッドは、config.time\_zone が設定されている場合に Time.zone.today を返し、そうでない場合には Date.today を返します。

#### ★ fragment\_exist? メソッド

cache\_store に新しい二つのメソッド fragment\_exist? と exist? が追加されました。

この fragment\_exist? メソッドは、ご想像の通り、与えられたキーの断片キャッシュ (“fragment cache”) が存在しているかを確認します。基本的には、有名な次の呼び出しの置き換えです。

---

```
read_fragment(path).nil?
```

---

fragment\_exist? メソッドがコントローラーで使われるヘルパーメソッドなのに対して、cache\_store にも exist? メソッドが追加されました。

#### ★ UTC なのか GMT なのか

興味深い修正です。今まで Rails は UTC という頭字語をたくさん使っていましたが、バージョン 2.1 では TimeZone.to\_s が GMT を返す様になりました。これは、エンドユーザーの間で GMT の方が一般的だからです。

Windows のコントロールパネルや、Google や Yahoo の製品を見ればわかると思いますが、それらも GMT という頭字語を利用しています。

```
-----  
TimeZone['Moscow'].to_s # => "(GMT+03:00) Moscow"  
-----
```

#### ★ json\_escape メソッド

json\_escape メソッドは html\_escape メソッドの様に動作します。ドキュメンテーションなどで、HTML 内に JSON の文字列を表示したい場合などに便利でしょう。

```
-----  
puts json_escape("is a > 0 & a < 10?")  
# => "is a ¥u003E 0 ¥u0026 a ¥u003c 10?"  
-----
```

メソッド j というショートカットも ERB 内で利用可能です。

```
-----  
<%= j @person.to_json %>  
-----
```

もし、すべての JSON コードを自動的にエスケープしたい場合は、environment.rb で以下の様に設定します。

```
-----  
ActiveSupport.escape_html_entities_in_json = true  
-----
```

#### ★ mem\_cache\_store のオプション

以前、ActiveSupport::Cache が Memcache-Client を取り込み、今までにないほど便利にはなりましたが、memcached サーバーの IP を変更する以外のオプションが提供されていませんでした。

しかし、ジョナサン・ヴァイズ (*Jonathan Weiss*) が書いたパッチによって、以下の様なオプションが Rails 2.1 では提供されます。

```
-----  
ActiveSupport::Cache.lookup_store :mem_cache_store, "localhost"  
-----
```

```
ActiveSupport::Cache.lookup_store :mem_cache_store,  
  "localhost", '192.168.1.1', :namespace => 'hoge'  
-----
```

もしくは、以下の様に設定することも可能です。

```
-----  
config.action_controller.fragment_cache_store = :mem_cache_store,  
  'localhost', { :compression => true,  
                :debug => true, :namespace => 'hoge' }  
-----
```

#### ★ Time.current メソッド

Time クラスの新しいメソッドです。current メソッドは、config.time\_zone が設定されている場合は Time.zone.now を、そうでない場合は Time.now を返します。

```
-----  
# config.time_zone の設定に合わせて値を返します  
Time.current  
-----
```

メソッド since と ago も同様に戻り値が変わり、config.time\_zone が設定されている場合は、TimeWithZone クラスのオブジェクトを返します。

他にも、datetime\_select、select\_datetime や、select\_time メソッドもデフォルトの値が Time.current のものに変更されました。

#### ★ 余計なスペースを削除する squish メソッド

String オブジェクトに新しく、squish と squish! メソッドが追加されました。このメソッドは、文字列の前後にある余白を削除する strip メソッドの様に動作しますが、文字列中の無用な二つ以上のスペースも削除します。以下の例を見てください。

```
-----  
"    A    text    full    of    spaces    ".strip  
# => "A    text    full    of    spaces"  
  
"    A    text    full    of    spaces    ".squish  
# => "A text full of spaces"  
-----
```

# 第3章

## ActiveResource

ActiveResource は、RESTful システムのクライアント実装を担当レイヤーです。ActiveResource を経由すると、外部の RESTful サービスをプロクシーの様に動作するオブジェクトを通じて利用することができます。

### ★ メールアドレスをユーザー名にできる

たまにメールアドレスをユーザー名として利用するサービスがありますが、その様なサービスは、以下の様な URL を利用することになります

```
-----  
http://hoge@example.com:password@api.restfulservice.com/  
-----
```

この様なアットマーク (“@”) が二つある URL を渡すと、パーサーがうまく分析できない為に問題が起きていました。解決策として ActiveResource の利用方法は若干拡張され、以下の様になり、メールアドレスを認証に利用することも楽になりました。

```
-----  
class Person < ActiveResource::Base  
  self.site = “http://api.resfulservice.com”  
  self.user = “hoge@example.com”  
  self.password = “password”  
end  
-----
```

### ★ clone メソッド

既存のリソースのコピーを取れる様になりました。

```
-----  
taro = Person.find(1)  
not_taro = ryan.clone  
not_taro.new? # => true  
-----
```

この clone メソッドは、クラス変数はコピーせず、リソース変数のみをコピーすることに気をつけてください。

```
-----
taro = Person.find(1)
taro.address = StreetAddress.find(1, :person_id => taro.id)
taro.hash = { :not => " ActiveRecord のインスタンス (ではない) " }

not_taro = taro.clone
not_taro.new?      # => true
not_taro.address  # => NoMethodError
not_taro.hash     # => { :not => "ActiveRecord のインスタンス (ではない) " }
-----
```

## ★ タイムアウト設定

`ActiveResource` は HTTP を使って RESTful API にアクセスしますが、レスポンスの悪いサーバーやダウンしているサーバーにアクセスすると、処理に影響がでてしまいました。ActiveResource の呼び出しがタイムアウトすることもあります。

Rails 2.1 では、API リクエストのタイムアウトを設定できるようになりました。

```
-----
class Person < ActiveResource::Base
  self.site = "http://api.people.com:3000/"
  self.timeout = 5 # 5秒間レスポンスが無ければタイムアウト
end
-----
```

この例では、タイムアウトを5秒に設定しています。タイムアウト時間を低く設定して、API リクエストに問題が生じた際の処理を速くし、連鎖的に問題が発生しない様にするのが推奨されています。万が一、連鎖的にこの問題が発生すると、サーバーの許容量が低下する可能性があります。

内部的に `ActiveResource` は `Net::HTTP` を利用して HTTP リクエストをしています。ActiveResource でタイムアウトの設定をした場合、同じ値が `Net::HTTP` オブジェクトの `read_timeout` にも設定されます。

デフォルトは60秒に設定されています。

# 第4章

## ActionPack

ActionPack は、HTML や XML、JavaScript 等、エンドユーザーに画面を生成する ActionView と、ビジネスフローを操作する ActionView で構成されています。

ウィキペディアより

### ★ タイムゾーン

#### タイムゾーンの設定

time\_zone\_select メソッドに新しいオプションが追加され、ユーザーが TimeZone を選択しなかった場合、またはデータベース項目が null の場合のデフォルトの TimeZone を設定できる様になりました。これを設定するためには、以下の様に :default オプションを設定します。

```
-----  
time_zone_select("user", "time_zone", nil, :include_blank => true)  
  
time_zone_select("user", "time_zone", nil,  
  :default => "Pacific Time (US & Canada)")  
time_zone_select("user", "time_zone", TimeZone.us_zones,  
  :default => "Pacific Time (US & Canada)")  
-----
```

:default オプションが設定されている場合は、選択された TimeZone と共に表示されるべきです。

#### formatted\_offset メソッド

Time と DateTime クラスに追加された formatted\_method は、UTC 時間との差異を "+HH:MM" フォーマットで返します。例えば、日本のタイムゾーンであれば、"+09:00" が返されます。

いくつか例を見てみましょう。

DateTime オブジェクトから UTC との差異時間を取得する。

```
-----
datetime = DateTime.civil(2000, 1, 1, 0, 0, 0, Rational(-6, 24))
datetime.formatted_offset      # => "-06:00"
datetime.formatted_offset(false) # => "-0600"
-----
```

Time オブジェクトからも同じように。

```
-----
Time.local(2000).formatted_offset      # => "-06:00"
Time.local(2000).formatted_offset(false) # => "-0600"
-----
```

引数に `false` を与えるとフォーマットされていない値の String オブジェクト、そうでない場合はフォーマットされた値の String オブジェクトが戻り値になります。

### **with\_env\_tz メソッド**

`with_env_tz` メソッドを利用すると、様々なタイムゾーンを利用したテストをととても簡単に書くことができます。

```
-----
def test_local_offset
  with_env_tz 'US/Eastern' do
    assert_equal Rational(-5, 24), DateTime.local_offset
  end
  with_env_tz 'US/Central' do
    assert_equal Rational(-6, 24), Datetime.local_offset
  end
end
-----
```

元々このヘルパーは、`with_timezone` という名前になるはずでしたが、`ENV['TZ']` と `Time.zone` のどちらのタイムゾーンを設定するのをはっきりさせるために、`with_env_tz` という名前になりました。

### **Time.zone\_reset! メソッド**

このメソッドは使われていない為、削除されました。

### **Time#in\_current\_time\_zone メソッド**

`Time.zone` が `nil` の時、`self` を返す様に変更されました。

### **Time#change\_time\_zone\_to\_current メソッド**

`Time.zone` が `nil` の時、`self` を返す様に変更されました。

## TimeZone#now メソッド

TimeZone#now メソッドは、Time.zone で設定されたタイムゾーンの現在の時間を ActiveSupport::TimeWithZone オブジェクトとして、例えば以下の様に返す様になりました。

```
-----  
Time.zone = 'Hawaii' # => "Hawaii"  
Time.zone.now       # => Wed, 23 Jan 2008 20:24:27 HST -10:00  
-----
```

## compare\_with\_coercion メソッド

Time と DateTime クラスに新しく compare\_with\_coercion メソッドと、そのエイリアスである <=> が作られました。これによって、Time と DateTime クラスおよび ActiveSupport::TimeWithZone インスタンスの間で、時間的比較ができるようになりました。理解を深めるために以下の例を見てみましょう。各行のコメント部分は比較結果になります。

```
-----  
Time.utc(2000) <=> Time.utc(1999, 12, 31, 23, 59, 59, 999) # 1  
Time.utc(2000) <=> Time.utc(2000, 1, 1, 0, 0, 0) # 0  
Time.utc(2000) <=> Time.utc(2000, 1, 1, 0, 0, 0, 001) # - 1  
  
Time.utc(2000) <=> DateTime.civil(1999, 12, 31, 23, 59, 59) # 1  
Time.utc(2000) <=> DateTime.civil(2000, 1, 1, 0, 0, 0) # 0  
Time.utc(2000) <=> DateTime.civil(2000, 1, 1, 0, 0, 1) # -1  
  
Time.utc(2000) <=> ActiveSupport::TimeWithZone.new(Time.utc(1999, 12,  
31, 23, 59, 59))  
Time.utc(2000) <=> ActiveSupport::TimeWithZone.new(Time.utc(2000, 1,  
1, 0, 0, 0))  
Time.utc(2000) <=> ActiveSupport::TimeWithZone.new(Time.utc(2000, 1,  
1, 0, 0, 1))  
-----
```

## TimeWithZone#between? メソッド

TimeWithZone クラスに、インスタンスが与えられた二つのインスタンスの時間的範囲内にあるかどうかを確認する between? メソッドが追加されました。以下が使用例です。

```
-----  
@twz.between?(Time.utc(1999,12,31,23,59,59), Time.utc(2000,1,1,0,0,1))  
-----
```

## TimeZone#parse メソッド

String をパースして、ActiveSupport::TimeWithZone インスタンスを作る parse メソッドが TimeZone クラスに追加されました。以下が使用例です。

```
-----  
Time.zone = "Hawaii"  
# => "Hawaii"  
Time.zone.parse('1999-12-31 14:00:00')  
# => Fri, 31 Dec 1999 14:00:00 HST -10:00  
  
Time.zone.now  
# => Fri, 31 Dec 1999 14:00:00 HST -10:00  
Time.zone.parse('22:30:00')  
# => Fri, 31 Dec 1999 22:30:00 HST -10:00  
-----
```

## TimeZone#at

このメソッドを使うと、UNIX epoch からの秒数を使って ActiveSupport::TimeWithZone インスタンスを作ることができます。

```
-----  
Time.zone = "Hawaii" # => "Hawaii"  
Time.utc(2000).to_f # => 946684800.0  
Time.zone.at(946684800.0)  
# => Fri, 31 Dec 1999, 14:00:00 HST -10:00  
-----
```

## その他のタイムゾーン関連メソッド

TimeWithZone クラスには他にも to\_a、to\_f、to\_i、httpdate、rfc2822、to\_yaml、to\_datetime や、eql? も追加されています。より詳しい情報は、Ruby on Rails の公式ドキュメントを参照してください。

## Ruby 1.9 に向けて

Ruby 1.9 では、Time クラスに、以下の様ないくつかの新しいメソッドが追加されます。

```
-----  
Time.now  
# => Thu Nov 03 18:58:25 CET 2005  
  
Time.now.sunday?  
# => false  
-----
```

各曜日に合わせて sunday?、monday?、tuesday? ... というメソッドが提供されます。

他にも興味深いのが、Time オブジェクトの to\_s メソッドの戻り値の変更です。現行のバージョンでは、Time.new.to\_s とすると、以下の様な結果が得られます。

```
-----  
Time.new.to_s  
# => "Thu Oct 12 10:39:27 +0200 2006"  
-----
```

一方 Ruby 1.9 では、以下の様な結果になります。

```
-----  
Time.new.to_s  
# => "2006-10-12 10:39:24 +0200"  
-----
```

Rails 2.1 とこれらの変更の関連性は为什么呢。Rails は、これらの変更に対応できるように準備が進められています。例えば、TimeWithZone クラスは、最初の例に対応する様に実装されています。

#### ★ 自動リンク auto\_link メソッド

知らない方もいるかもしれませんが、auto\_link メソッドはどんな文章でも引数として受け取り、返します。もし、URL やメールアドレスが含まれている場合は、ハイパーリンク付きの文章を返します。以下の例を見てみましょう。

```
-----  
auto_link("次のサイトを開いてください。 http://www.rubyonrails.com")  
# => 次のサイトを開いてください。 http://www.rubyonrails.com  
-----
```

しかし、アマゾンなどの、URL 中にイコール "=" が使われているものを渡すと、このメソッドは URL を正しく認識しませんでした。その様な場合のこのメソッドの挙動を見てみましょう。

```
-----  
auto_link("http://www.amazon.com/Testing/ref=pd\_bbs\_sr\_1")  
# => http://www.amazon.com/Testing/ref  
-----
```

イコールの直前でメソッドが終了しているのがわかるでしょうか。このメソッドは、イコールの入った URL をサポートしていないのです。いや、正確に言えば、サポートしていません。Rails 2.1 では修正されています。

この auto\_link メソッドはその後、以下の例の様な、括弧付きの URL にも対応しました。

---

[http://en.wikipedia.org/wiki/Sprite\\_\(computer\\_graphics\)](http://en.wikipedia.org/wiki/Sprite_(computer_graphics))

---

### ★ form\_for メソッドのラベル

Scaffold を使ってフォームを作ると以下の様なコードが生成されます。

---

```
<% form_for(@post) do |f| %>
  <p>
    <%= f.label :title %><br />
    <%= f.text_field :title %>
  </p>
  <p>
    <%= f.label :body %><br />
    <%= f.text_area :body %>
  </p>
  <p>
    <%= f.submit "Update" %>
  </p>
<% end %>
```

---

この様に、理にかなったコードを書くための label メソッドが form\_for メソッドのブロック引数に追加されました。このメソッドは、項目名を HTML の <label> タグに囲んで表示します。

---

```
>> f.label :title
=> <label for="post_title">Title</label>

>> f.label :title, "A short title"
=> <label for="post_title">A short title</label>

>> f.label :title, "A short title", :class => "title_label"
=> <label for="post_title" class="title_label">A short title</label>
```

---

<label> タグ中の for 属性に注目してください。“post\_title” は、投稿の件名を入力するテキストボックスの名前です。この <label> タグは HTML 中の post\_title と関連付けられていて、ユーザーがラベルをクリックすると、関連付けられたコントロール（この場合はテキストボックス）がフォーカスを取得します。

ロビー・ラッセル (“*Robby Russell*”) がこれについて興味深い次の投稿をブログにしています。<http://www.robbyonrails.com/articles/2007/12/02/that-checkbox-needs-a-label>

FormTagHelper にも `label_tag` メソッドが追加されました。このメソッドは前に紹介した `label` メソッドに似ていますが、もっと単純で、以下の様に動作します。

```
-----  
>> label_tag 'name'  
=> <label for="name">Name</label>  
  
>> label_tag 'name', 'お名前'  
=> <label for="name">お名前</label>  
  
>> label_tag 'name', nil, :class => 'small_label'  
=> <label for="name" class="small_label">Name</label>  
-----
```

また、このメソッドは `for` オプションも受け取ります。例を見てみましょう。

```
-----  
label(:post, :title, nil, :for => "my_for")  
# => <label for="my_for">Title</label>  
-----
```

#### ★ 部分テンプレート呼び出し

Rails でのソフトウェア開発で一般的なものは、部分テンプレート (“partials”) を利用して、コードの重複を避けることです。例を見てみましょう。

```
-----  
<% form_for :user, :url => users_path do %>  
  <%= render :partial => 'form' %>  
  <%= submit_tag => 'Create' %>  
<% end %>  
-----
```

部分テンプレートはコードフラグメント（もしくはテンプレートとも）です。部分テンプレートを利用することの利点は、コードの不要な重複を避けることです。使い方は簡単で、`render :partial => “name”` という風に呼び出し、`:partial` プロパティに対応するファイル名のテンプレートを作成するだけです。ただし、そのファイル名には、部分テンプレートであることを表すアンダースコアを頭に付けてください。`:partial` が “name” であれば、`_name` といった風입니다。

上の例で示したコードは今までの使い方、新しいバージョンでは、以下の様に同じことを別の方法で行います。

```
-----  
<%= form_for(@user) do |f| %>  
  <%= render :partial => f %>  
  <%= submit_tag 'Create' %>  
<% end %>  
-----
```

この方法では、“users/\_form” という部分テンプレートが、FormBuilder によって生成された form という変数付きで呼び出されます。

今までの呼び出し方も使用可能です。

#### ★ ATOM フィードの新しいネームスペース

atom\_feed というメソッドを知っていますか。このメソッドは Rails 2.0 からの新機能で、簡単に ATOM フィードを生成するメソッドです。使用例を見てみましょう。

```
-----  
atom_feed do |feed|  
  
  feed.title("My Blog")  
  feed.updated(@posts.first.created_at)  
  
  for post in @posts  
    feed.entry(post) do |entry|  
      entry.title(post.title)  
      entry.content(post.body, :type => 'html')  
      entry.author do |author|  
        author.name("Your Name")  
      end  
    end  
  end  
end  
  
end  
-----
```

ATOM フィードが何か知らない方の為に説明しておく、ATOM とは、ブログやニュースサイトなど頻りに更新されるコンテンツのメタ情報（見出しや要約など）をインターネット上で公開する為の protocol です。ATOM フィードは、XML を利用して書かれ、メディアタイプは application/atom+xml と規定されています。

Rails 2.0 の最初のバージョンでは、:language、:root\_url あと :url というオプションを受け取りました。これについての詳細は Rails の公式ドキュメントを参照してください。Rails 2.1 では、これらのメソッドに加えて、新しくフィードのルートに新しいネームスペースを含めることができるようになりました。

例えば...

```
-----  
atom_feed('xmlns:app' => 'http://www.w3.org/2007/app') do |feed|  
-----
```

というコードは、下の様な XML を返します。

```
-----  
<feed xml:lang="en-US" xmlns="http://www.w3.org/2005/Atom"  
  xmlns:app="http://www.w3.org/2007/app">  
-----
```

変更を加えて、この様にもできます。

```
-----  
atom_feed(  
  {  
    'xmlns:app' => 'http://www.w3.org/2007/app',  
    'xmlns:openSearch' => 'http://a9.com/-/spec/opensearch/1.1/'  
  }  
) do |feed|  
  
  feed.title("My Blog")  
  feed.updated(@posts.first.created_at)  
  feed.tag!(openSearch:totalResults, 10)  
  
  for post in @posts  
    feed.entry(post) do |entry|  
      entry.title(post.title)  
      entry.content(post.body, :type => 'html')  
      entry.tag!('app:edited', Time.now)  
  
      entry.author do |author|  
        author.name("Your Name")  
      end  
    end  
  end  
end  
  
end  
-----
```

#### ★ キャッシュ

すべての `fragment_cache_key` メソッドがデフォルトでプレフィックス “view/” を付けたネームスペースを返す様になりました。

ActionController::Caching::Fragments::\* からすべてのキャッシュストアが削除され、それらが ActiveSupport::Cache::\* に移動されました。もし、 ActionController::Caching::Fragments::MemoryStore の様に、キャッシュストアへの参照をしているのならば、 ActiveSupport::Cache::MemoryStore に変更する必要があります。

また、 ActionController::Base.fragment\_cache\_store が削除され、代わりに ActionController::Base.cache\_store が追加されました。

一方 ActiveRecord::Base には cache\_key というメソッドも追加されました。このメソッドは、新しいライブラリである ActiveSupport::Cache::\* を利用して ActiveRecord のオブジェクトをキャッシュするのに使われ、以下の様に動作します。

```
-----  
>> Product.new.cache_key  
=> "products/new"  
  
>> Product.find(5).cache_key  
=> "products/5"  
  
>> Person.find(5).cache_key  
=> "people/5-20071224150000"  
-----
```

ActiveSupport::Gzip.compress と decompress という、Zlib を簡単に使えるインターフェースも提供されました。

キャッシュストアの指定するには、環境設定内の config.cache\_store 変数を変更します。特筆すべきは、tmp/cache ディレクトリが存在する場合、デフォルトで FileStore に設定され、それ以外の場合は MemoryStore に設定されます。キャッシュストアの指定は次の様に行います。

```
-----  
config.cache_store = :memory_store,  
config.cache_store = :file_store, "/path/to/cache/directory",  
config.cache_store = :drb_store, "druby://localhost:9192"  
config.cache_store = :mem_cache_store, "localhost"  
config.cache_store = MyOwnStore.new("parameter")  
-----
```

もっとわかりやすくする為に、以下の様なコメントも environments/production.rb に追加されています。

```
-----  
# Use a different cache store in production  
# config.cache_store :mem_cache_store  
-----
```

#### ★ String.titleize メソッドのバグ

文字列をタイトルとして適切なフォーマットにする String クラスの titleize メソッドには、以下の様に ”s”（アポストロフィー後のs）を大文字にしてしまうバグがありました。

```
-----  
>> "hama's blog".titleize  
=> "Hama'S Blog"  
-----
```

このバグは修正され、以下の様に適切な文字列を返す様になりました。

```
-----  
>> "hama's blog".titleize  
=> "Hama's Blog"  
-----
```

#### ★ action\_name メソッド

ビュー内で、どのアクションが呼ばれたのかを、以下の様に取得できる action\_name メソッドが追加されました。

```
-----  
<%= action_name %>  
-----
```

戻り値は、params[:action] と同じ内容のものですが、より上品に整形されています。

#### ★ 条件付き caches\_action メソッド

caches\_action メソッドが :if オプションを受け取る様になり、以下の例の様に、アクションがキャッシュされる条件を指定できるようになりました。

```
-----  
caches_action :index, :if => Proc.new { |c| !c.request.format.json? }  
-----
```

この例では、JSON のリクエストでない場合のみ index アクションがキャッシュされます。

★ 条件付き `cache_page` メソッド

`cache_action` メソッドと同じように、`cache_page` メソッドも

以下の例の様に、条件を受け取る様になりました。

```
-----  
# Rails 2.0 での使い方  
cache_page :index  
  
# Rails 2.1 では :if オプションを受け取る様になりました。  
cache_page :index, :if => Proc.new { |c| !c.request.format.json? }  
-----
```

★ テストできる様になったフラッシュメッセージ

この、フラッシュメッセージをテストできない問題に悩まされていた人はたくさんいたと思います。この問題は、Rails がテストスクリプトに進む前に `flash` に保存されていたメッセージを消してしまう為に起きていました。

Rails 2.1 では、この問題が解決され、以下の様にテストできるようになりました。

```
-----  
assert_equal '>value_now<', flash['test_now']  
-----
```

★ ビュー外でも利用できる様になったヘルパー

ヘルパーメソッドがコントローラーでも使えたらと幾度となく思ったことでしょう。今までこれを可能にする為には、ヘルパーのモジュールをコントローラーに読み込ませる必要がありました。コードが汚くなってしまいました。

Rails 2.1 では、以下の様に簡単に使える、ビュー外でもヘルパーを呼び出せるプロクシーが用意されました。

```
-----  
# 例えばヘルパーの simple_format メソッドを利用するには  
ApplicationController.helpers.simple_format(text)  
-----
```

単純明快です。

## ★ JSON を利用した POST リクエスト

新しい Rails は、HTTP POST リクエストで JSON を受け取れるようになりました。以下の様な POST リクエストができます。

```
-----  
POST /posts  
{ "post": { "title": "Breaking News" } }  
-----
```

送られた変数はすべて params に格納されます。以下の様なコードが可能です。

```
-----  
def create  
  @post = Post.create params[:post]  
  # ...  
end  
-----
```

## ★ パス名

私のブログ (<http://www.nomedojogo.com>) を読んでいる人は、Custom Resource Name プラグインを知っているでしょうが、このプラグインもそろそろ不要になりそうです。

以前より、:as オプションを利用してルートの名前を変えることができましたが（これは私のプラグインで互換性を保つ為に実装したものでもありますが、）新しいバージョンでは、:path\_names オプションも提供され、アクションの名前を変えることができるようになりました。

```
-----  
map.resource :schools, :as => 'gakkou',  
               :path_names => { :new => 'shin' }  
-----
```

## ★ ルートファイルの場所指定

Rails 2.1 では、environments.rb 内で次の変数を設定することにより、ルートファイル routes.rb の場所を指定できるようになりました。

```
-----  
config.routes_configuration_file  
-----
```

この機能は、同じモジュールやプラグインを共有する二つのフロントエンドがある場合などに便利でしょう。

例えば、`getstisfaction.com` と `api.getstisfaction.com` という二つのフロントエンドがモデルのみを共有していて、`getstisfaction.com` では、検索エンジン最適化の為に独自のルートを利用し、一方 `api.getstisfaction.com` では標準的なルートを利用する場合などに便利です。

#### ★ session メソッドの `:on` と `:off` 引数

もう知らないことが多いかもしれませんが、Rails では以下の様にしてセッション機能を無効にできます。

```
-----  
class ApplicationController < ActionController::Base  
  session :off  
end  
-----
```

特定のコントローラーのセッション機能を無効にすることもできますが、この例では、すべてのコントローラーのセッション機能を無効にしています。

では、この様にすべてのコントローラーのセッション機能を無効にした時、特定のコントローラーのみ有効にするにはどうしたら良いでしょうか。Rails 2.1 では、`session` メソッドが `:on` オプションを受け取る様になり、以下の様にこれが可能になりました。

```
-----  
class UsersController < ApplicationController  
  session :on  
end  
-----
```

#### ★ 簡単になったヘルパーのテスト

今までのバージョンでは、ヘルパーのテストをするのが非常に面倒でした。100% のカバレッジを確保する為に、どれほどの時間を時間を費やしたかわかりません。

Rails 2.1 では、`ActionView::TestCase` クラスが提供され、ヘルパーのテストが簡単になりました。まず、今までのテストを見てみましょう。

```
-----  
module PeopleHelper  
  def title(text)  
    content_tag(:h1, text)  
  end  
  
  def homepage_path  
    people_path  
  end  
end  
-----
```

```
end
end
```

---

次に Rails 2.1 での同じテストを見てみましょう。

---

```
class PeopleHelperTest < ActionController::TestCase
  def setup
    ActionController::Routing::Routes.draw do |map|
      map.people 'people', :controller => 'people', :action => 'index'
      map.connect ';/controller/:action/:id'
    end
  end

  def test_title
    assert_equal "<h1>Ruby on Rails</h1>", title("Ruby on Rails")
  end

  def test_homepage_path
    assert_equal "/people", homepage_path
  end
end
```

---

# 第5章

## ActionController

ActionController は、ユーザーからの HTTP リクエストを取得し、何を実行して、どのビューを表示すべきかや、どのアクションにリダイレクトすべきか等を判断するレイヤーです。アクションは、コントローラー内の public メソッドとして定義され、ルートを通じて自動的に提供されます。

★ ActionController::Routing モジュール

### map.root メソッド

map.root メソッドでエイリアスを利用して、さらに DRY (*“Don't Repeat Yourself”*) にできる様になりました。

今までのバージョンでは、以下の様な方法が一般的でしたが、

```
-----  
map.new_session :controller => 'sessions', :action => 'new'  
map.root :controller => 'sessions', :action => 'new'  
-----
```

Rails 2.1 では、以下の様にできます。

```
-----  
map.new_session :controller => 'sessions', :action => 'new'  
map.root :new_session  
-----
```

### Rails によるルートの認識と構築

Rails がルートを構築する際には、今まで Rails は、すべてのルートを洗い出し、一つ一つ認識していたので、多くの場合、あまり高速ではありませんでした。新しいより賢い実装では、ルートはツリー構造で構築され、プレフィックス化と類似ルートの省略が行われ、約 2.7 倍ルートの認識は高速化されました。

### assert\_routing テストメソッド

assert\_routing メソッドを利用することで、HTTP を利用したルートのテストを行える様になりました。以下の例を見てみましょう。

```
-----
assert_routing(
  { :method => 'put',
    :path   => '/product/321' },
  { :controller => 'product',
    :action   => 'update',
    :id      => '321' }
)
-----
```

## map.resources メソッド

仮に、英語以外の言語で書かれたウェブサービスを構築していて、ルートをその言語にしたいとしましょう。例えば、以下の様なルートではなく、

```
-----
http://mysite.jp/products/1234/reviews
-----
```

以下の様なルートにしたいとします。

```
-----
http://mysite.jp/shohin/1234/hyoka
-----
```

新しいバージョンでは、map.resources メソッドに :as オプションが提供され、ルート名前を変更できるようになり、この様なルートの作成が可能になりました。上の例の実装をして、ローマ字のルートを作ってみましょう。

```
-----
map.resources :products, :as => 'shohin' do |product|
  # product_reviews_path(product) == '/shohin/1234/hyoka'
  product.resources :product_reviews, :as => 'hyoka'
end
-----
```

## ★ ActionController::Caching::Sweeping モジュール

今までの Rails でキャッシュスイーパーを定義するには、シンボルを使ってクラスに知らせる必要がありました。

```
-----
class ListsController < ApplicationController
  caches_action :index, :show, :public, :feed
  cache_sweeper :list_sweeper, :only => [ :edit, :destroy, :share ]
end
-----
```

新しいバージョンでは、明示的にクラスを指定して定義することが可能になりました。例えば以下の様に、指定したいスイーパーがモジュールに含まれている場合などに必要になります。もちろん、今まで通りのシンボルを使った定義の仕方も可能です。

```
-----  
class ListsController < ApplicationController  
  caches_action :index, :show, :public, :feed  
  cache_sweeper OpenBar::Sweeper, :only => [ :edit, :destroy, :share ]  
end  
-----
```

# 第6章

## ActionView

ActionView は、ERB で書かれたテンプレートを変換し、ユーザーに視覚的インターフェースを提供するレイヤーです。

★ ActionView::Helpers::FormHelper モジュール

### field\_for と form\_for メソッドの :index オプション

fields\_for と form\_for メソッドに :index オプションが追加され、また、それぞれのオブジェクトに :index => nil と設定する必要がなくなりました。以下の例を見てみましょう。

今まではこの様なコードでした。

```
-----  
<% fields_for "projects[talk_attributes][]", task do |f| %>  
  <%= f.text_field :name, :index => nil %>  
  <%= f.hidden_field :id, :index => nil %>  
  <%= f.hidden_field :should_destroy, :index => nil %>  
<% end %>  
-----
```

新しいバージョンでは同じコードが以下の様にできます。

```
-----  
<% fields_for "project[task_attributes][]", task,  
  :index => nil do |f| %>  
  <%= f.text_field :name %>  
  <%= f.hidden_field :id %>  
  <%= f.hidden_field :should_destroy %>  
<% end %>  
-----
```

★ ActionView::Helpers::DateHelper モジュール

DateHelper モジュール中の日付に関連するすべてのメソッド、例えば date\_select、time\_select や select\_datetime 等が、HTML オプションを受け取るようになりました。date\_select の例を見てみましょう。

```
-----  
<%= date_select 'item', 'happening', :order => [:day],  
          :class => 'hoge' %>  
-----
```

## date\_helper メソッド

date\_helper メソッドは初期値が Date.current に変更されました。

### ★ ActionView::Helpers::AssetTagHelper

## register\_javascript\_expansion メソッド

このメソッドは、javascript\_include\_tag メソッドで指定されたシンボルが渡された際に読み込まれる JavaScript ファイルをあらかじめ登録しておく為に使います。これは、JavaScript を public/javascripts にインストールする様なプラグインの init.rb で、読み込ませたい JavaScript をあらかじめ登録しておく為に考えられました。どの様に動くのかを見てみましょう。

```
-----  
# プラグインの init.rb 中  
ActionView::Helpers::AssetTagHelper.register_javascript_expansion  
  :monkey => ["head", "body", "tail"]  
  
# ビュー中  
javascript_include_tag :monkey  
  
# 以下の様な HTML が生成されます。  
<script type="text/javascript" src="/javascripts/head.js"></script>  
<script type="text/javascript" src="/javascripts/body.js"></script>  
<script type="text/javascript" src="/javascripts/tail.js"></script>  
-----
```

## register\_stylesheet\_expansion メソッド

このメソッドは、上の register\_javascript\_expansion のスタイルシート版で、後に stylesheet\_link\_tag で指定したシンボルが渡された際に読み込まれるスタイルシートをあらかじめ登録しておく為に使います。例を見てみましょう。

```
-----  
# プラグインの init.rb 中  
ActionView::Helpers::AssetTagHelper.register_stylesheet_expansion  
  :monkey => ["head", "body", "tail"]  
  
# ビュー中  
stylesheet_link_tag :monkey  
-----
```

# 以下の様な HTML が生成されます。

```
<link href="/stylesheets/head.css" media="screen" rel="stylesheet"
type="text/css" />
<link href="/stylesheets/body.css" media="screen" rel="stylesheet"
type="text/css" />
<link href="/stylesheets/tail.css" media="screen" rel="stylesheet"
type="text/css" />
```

---

★ ActionView::Helpers::FormTagHelper

**submit\_tag メソッド**

:confirm オプションが submit\_tag メソッドに追加されました。これは、link\_to のものと同じように動作します。例を見てみましょう。

```
submit_tag('Save changes', :confirm => 'Are you sure?')
```

---

★ ActionView::Helpers::NumberHelper

**number\_to\_currency メソッド**

number\_to\_currency メソッドが、:format オプションを受け取る様になり、戻り値のフォーマットを指定できるようになりました。以下の例を見てみましょう。

```
# JPY という単位で表示したい場合
number_to_currency(9.99, :separator => ",", :delimiter => ".", :unit
=> "JPY")
# => "JPY9.99"

# JPY の後にスペースが必要なので、:format オプションで指定
number_to_currency(9.99, :format => "%u %n", :separator =>
",", :delimiter => ".", :unit => "JPY")
# => "JPY 9.99"
```

---

以下の様に違ったカスタマイズも可能です。

```
number_to_currency(1980, :format => "%n%u", :unit => "円")
```

---

:format オプションに利用可能なパラメータには、%u の単位と、%n の数値があります。

## ★ ActionView::Helpers::TextHelper

### excerpt メソッド

以前から提供されている `excerpt` メソッドは、文字列から指定した文字列を検索し、その文字列と、前後の文字列を指定した分だけ集め、必要ならば省略されたことを示す “...” を付加します。例を見てみましょう。

```
-----  
excerpt('This is an example', 'an', 5)  
# => "...s is an examp..."  
-----
```

しかし、このメソッドにはバグがあります。数えてみるとわかりますが、`an` の後ろ側が、例で指定した5文字ではなく、6文字が返されています。この問題は、新しいバージョンで修正されました。修正されたメソッドを使用した例を見てみましょう。

```
-----  
excerpt('This is an example', 'an', 5)  
# => "...s is an exam..."  
-----
```

### simple\_format メソッド

`simple_format` メソッドは、与えられた文字列を HTML にフォーマットして表示するメソッドです。文字列の改行 `\n` は HTML タグの `<br />` に置き換わり、二つ改行が続くと `<p>` タグを使って文字列を分けます。

Rails 2.1 では、このメソッドにオプションが追加されました。文字列だけでなく、`<p>` タグが使用された場合の、`<p>` タグのプロパティを設定できるようになりました。例を見てみましょう。

```
-----  
simple_format("Hi mom!", :class => "description")  
# => "<p class=\"description\">Hi mom!</p>"  
-----
```

プロパティは、メソッドで作られたすべての `<p>` タグに適用されます。

# 第7章

## Railties

### ★ config.gem メソッド

新しいバージョンでは、アプリケーションに必要な gem を設定しておき、まとめてインストールできるようにする為の config.gem メソッドが追加されました。設定は、environment.rb 中で以下の例の様にいきます。

```
-----  
config.gem "bj"  
config.gem "hpricot", :version => '0.6',  
                    :source  => "http://code.whytheluckystiff.net"  
config.gem "aws-s3", :lib => "aws/s3"  
-----
```

設定された gem をまとめてインストールするには、以下の Rake タスクを呼び出します。

```
-----  
# 設定された gem をすべてインストールする  
rake gems:install  
-----
```

設定された gem の一覧を表示するには、以下のタスクを呼び出します。

```
-----  
# 依存している gem の一覧を表示する  
rake gems  
-----
```

設定された gem が rails/init.rb を持っているなどの理由で、gem を vendor/gems に展開したい場合は以下の様にタスクを呼び出して、展開することができます。

```
-----  
# 指定した gem を vendor/gems/gem_name-x.x.x に展開する  
rake gems:unpack GEM=gem_name  
-----
```

呼び出すと、指定された gem が vendor/gems/gem\_name-x.x.x に展開されます。gem 名を指定しなかった場合は、すべての gem が vendor/gems 内に展開されます。

★ プラグインの config.gem メソッド

プラグインでもまた、config.gem の機能を利用できます。

Rails 2.0 までは、プラグインの init.rb は以下の様でした。

```
-----  
# open_id_authentication の init.rb  
require 'yadis'  
require 'openid'  
ActionController::Base.send :include, OpenIdAuthentication  
-----
```

Rails 2.1 では以下の様になります。

```
-----  
config.gem "ruby-openid", :lib => "openid", :version => "1.1.4"  
config.gem "ruby-yadis", :lib => "yadis", :version => "0.3.4"  
  
config.after_initialize do  
  ActionController::Base.send :include, OpenIdAuthentication  
end  
-----
```

このプラグインがインストールされた状態で、先ほどの設定された依存している gem をまとめてインストールする Rake タスクを実行すると、プラグインで設定されたこれらの gem もインストールされます。

★ gems:build タスク

gems:unpack タスクを通じて展開された gem のネイティブエクステンションをコンパイルする為の、gems:build タスクも追加されました。このタスクは以下の様に呼び出します。

```
-----  
rake gems:build # すべての gem のネイティブエクステンションをコンパイル  
rake gems:build GEM=mygem # 指定した gem のみ  
-----
```

★ サーバー起動時のメッセージ

Rails のサーバーが起動される際のメッセージに若干の改良が加えられ、読み込まれた Rails のバージョンが表示される様になりました。

```
-----  
Rails 2.1 application starting on http://0.0.0.0:3000  
-----
```

★ Rails.public\_path 変数

“public” ディレクトリーへのショートカットを提供する public\_path 変数が、Rails モジュールに追加されました。

```
-----  
Rails.public_path  
# => "/Users/hama/test_project/public"  
-----
```

★ Rails.logger、Rails.root、Rails.env と Rails.cache

Rails 2.1 では、RAILS\_DEFAULT\_LOGGER や、RAILS\_ROOT、RAILS\_ENV、RAILS\_CACHE という定数の代わりに、Rails モジュールで提供される、それぞれに対応する変数を利用します。

```
-----  
# RAILS_DEFAULT_LOGGER  
Rails.logger  
  
# RAILS_ROOT  
Rails.root  
  
# RAILS_ENV  
Rails.env  
  
# RAILS_CACHE  
Rails.cache  
-----
```

★ Rails.version 変数

今までのバージョンで、Rails のバージョンを実行時に知るためには、以下の様にしましたが、

```
-----  
Rails::VERSION::STRING  
-----
```

Rails 2.1 では、以下の様に変更されました。

```
-----  
Rails.version  
-----
```

★ プラグインの情報を取得する

Rails 2.0 では、プラグインの情報を得る機能が追加されましたが、多くの人が恐らく使ったことはないでしょう。しかし、プラグインのバージョンを知りたい時などに便利かもしれません。

この機能を試してみる為には、プラグインのディレクトリーに `about.yml` という、以下の様な内容のファイルを作ります。

```
-----  
author: Taro Yamada  
version: 1.0.0  
description: A description about the plugin  
url: http://yoidore.org  
-----
```

すると、以下の様な方法でプラグインの情報が取得できます。

```
-----  
plugin = Rails::Plugin.new(plugin_directory)  
plugin.about["author"] # => "Taro Yamada"  
plugin.about["url"]   # => "http://yoidore.org"  
-----
```

この機能の便利な利用法を思いついた方がいましたら、是非教えてください。もしかしたら、私が気がついていないだけなのかもしれません。

# 第8章

## Rake タスク、プラグイン、スクリプト

### ★ Rake タスク

#### **rails:update** タスク

新しいバージョンでは、`rake rails:freeze:edge` タスクを実行すると、自動的に `rails:update` タスクも実行され、設定ファイルと JavaScript ファイルの更新が行われます。

#### 127.0.0.1 のデータベース

`databases.rake` ファイルが更新され、今まではローカルにあるデータベースについては `localhost` だけを参照していましたが、新しいバージョンでは 127.0.0.1 も `create` と `drop` タスクで参照する様になりました。`databases.rake` ファイルはリファクタリングも施され、不要なコードの繰り返しが減りました。

#### 指定したバージョンの Rails を固める

Rails 2.1 以前は、指定したリビジョンの Rails をプロジェクト内に固めることができませんでしたが、そこでバージョンを指定することはできませんでした。Rails 2.1 では、以下の様なコマンドで、これが可能になりました。

```
-----  
rake rails:freeze:edge RELEASE=1.2.0  
-----
```

### ★ タイムゾーン

#### **time:zones:all** タスク

Rails が認識しているすべてのタイムゾーンを、基準時からの差異でグループ化して表示します。任意のオプション `OFFSET` を、例えば `OFFSET=+9` という様に、指定することで、特定のエリアだけを抽出することもできます。

#### **time:zones:us** タスク

アメリカ合衆国内のすべてのタイムゾーン表示します。`OFFSET` オプションも有効です。

#### **time:zones:local** タスク

OS に設定されているタイムゾーンで Rails が認識しているすべてのタイムゾーンを表示します。

## ★ スクリプト

### plugin スクリプト

script/plugin install コマンドに `-e`、`--export` オプションが追加されました。これを指定すると、`svn export` を利用してプラグインがインストールされます。また、plugin スクリプトは、git リポジトリにも対応しました。

### dbconsole スクリプト

このスクリプトは script/console のデータベース版です。実行すると、自動的にデータベースのコマンドラインが開かれます。

コードを見れば明確だと思いますが、現状では `mysql` と `postgresql`、そして `sqlite3` にしか対応していません。database.yml でそれら以外のデータベースが指定されている場合は、`"not supported for this database type"` というメッセージが表示されます。

## ★ プラグイン

### gem のままプラグインをインストール

rails/init.rb が含まれる gem がプロジェクトの vendor ディレクトリにある場合、プラグインとして認識され、Rails 起動時に読み込まれる様になりました。

### 標準パス外プラグインのジェネレーター

標準パス vendor/plugins 外にあるプラグインを Rails に読み込ませることは、以前より、以下の様なコードを environment.rb に含めることで可能でした。

```
-----  
config.plugin_paths = ['lib/plugins', 'vendor/plugins']  
-----
```

しかし、Rails 2.0 のバグで、この様な方法で読み込まれた標準パス外のプラグインに含まれるジェネレーターが、正しく認識されていませんでした。このバグは Rails 2.1 で潰されました。

# 第9章

## prototype.js と script.aculo.us

Rails 2.1 では、prototype.js 1.6.0.1 が、script.aculo.us 1.8.1 に移行するための準備として、使用されています。

# 第10章

## Ruby 1.9 に向けて

### ★ 詳細

Rails の更新の主な目的には、Ruby 1.9 への対応がありました。その為、細かい点も吟味され、次の Ruby に向けた互換性向上が図られました。ここでは、細かい点、例えば `File.exists?` メソッドが `File.exist?` に変更されたなど、は省いてあります。

また、Ruby 1.9 では Base64 (`base64.rb`) モジュールが標準ライブラリから外された為、Base64 の参照はすべて `ActiveSupport::Base64` へ置き換えられました。

### ★ DateTime クラスの新しいメソッド

Time クラスとの互換性を保つために、DateTime クラスに三つの新しいメソッドが追加されました。以下の例の `utc`、`utc?` と、`utc_offset` メソッドです。

```
-----  
>> date = DateTime.civil(2005, 2, 21, 10, 11, 12, Rational(-6, 24))  
# => Mon, 21 Feb 2005 10:11:12 -0600  
  
>> date.utc  
# => Mon, 21 Feb 2005 16:11:12 +0000  
  
>> DateTime.civil(2005, 2, 21, 10, 11, 12, Rational(-6, 24)).utc?  
# => false  
  
>> DateTime.civil(2005, 2, 21, 10, 11, 12, 0).utc?  
# => true  
  
>> DateTime.civil(2005, 2, 21, 10, 11, 12, Rational(-6,  
24)).utc_offset  
# => -21600  
-----
```

# 第11章

## デバッグ

### ★ ruby-debug の標準化

Rails のテストにおいて、再度 `ruby-debug` が標準で利用できる様になりました。`ruby-debug` の `gem` が既にインストールされていれば、`debugger` メソッドをコード中におくだけでデバッグが可能です。

# 第12章

## バグの修正

### ★ PostgreSQL 利用時の add\_column メソッド

PostgreSQL 利用時に、マイグレーションで既存のテーブルに項目を追加する際に問題となるバグが発生していました。以下のバグの例を見てみましょう。

この例のファイルは db/migrate/002\_add\_cost.rb です。

```
-----  
class AddCost < ActiveRecord::Migration  
  def self.up  
    add_column :items, :cost, :decimal, :precision => 6, :scale => 2  
  end  
  
  def self.down  
    remove_colun :items, :cost  
  end  
end  
-----
```

この例では、:precision => 6 と、:scale => 2 と設定された項目を作成しようとしているのが判ると思います。しかし、実際に db:migrate タスクを呼び出して作られるテーブルは以下の様になっていました。

項目	型	属性
id	integer	not null
descr	character varying(255)	
price	numeric(5,2)	
cost	numeric	

cost 項目に注目してください。一般的な numeric になっているのが判るでしょうか。本来は、price の様に、numeric(6,2) であるべきなのです。このバグが Rails 2.1 では修正され、項目が正しく作られる様になりました。

## ★ MIME タイプ

`request.format` にシンボルを渡してプロパティを設定できないバグがありましたが、これは修正されました。これによって、以下の様なコードが有効になりました。

```
-----  
request.format = :iphone  
assert_equal :iphone, request.format  
-----
```

## ★ `change_column` メソッド

`change_column` メソッドを利用して、マイグレーションで `:null =. false` として作成された項目を `:null => true` に変更しようとしたにも関わらず、実際には変更が適用されないバグが修正されました。

# 第13章

## 追加情報

### ★ クロスサイトスクリプティング対策

Rails 2.0 では、`application.rb` ファイルは以下の様な内容でした。

```
-----  
class ApplicationController < ActionController::Base  
  helper :all  
  protect_from_forgery  
end  
-----
```

`protect_from_forgery` というメソッドが呼ばれていることに注目してください。

クロスサイトスクリプティングをご存じない方の為に説明すると、これは、世の中のウェブサイトやウェブアプリケーションで最も多く見つかるセキュリティー問題で、悪意のあるユーザー（例えば社会意識のかけらもない十代の子供達）が、サイトを改ざんしたり、フィッシング詐欺を行ったり、サイトの JavaScript を乗っ取り、他のユーザーに対して任意のコードを実行させたりすることを可能にしていまう問題です。特に最後の攻撃は、クロスサイトリクエストフォージェリー (*CSRF: "Cross-site Request Forgery"*) と呼ばれています。

クロスサイトリクエストフォージェリーは、一般のユーザーが気づかぬうちに任意のコードを実行させるもので、AJAX の多用される昨今では、更に深刻な問題となっています。

`protect_from_forgery` メソッドは、アプリケーション中で使用されているすべてのフォームのリクエストが、アプリケーション自身からのものであって、他のサイトからのものではないと確認する為に使われるメソッドです。確認は、すべてのフォームと AJAX リクエストに含められるセッション情報を元にしたトークンを、リクエストを取得した際にコントローラーが有効であるかどうかを判断して行われます。

GET リクエストにはこの確認が行われません。というのも、GET リクエストが、データベースにデータを保存したり変更したりせず、データを呼び出すことだけに使われていれば、問題にはならないからです。

クロスサイトリクエストフォージェリーについてのより詳しい情報は、以下のアドレスを参照してください。

- <http://www.nomedojogo.com/2008/01/14/como-um-garoto-chamado-samy-pode-derrubar-seu-site/isc.sans.org/diary.html?storyid=1750>

しかし、この方法がクロスサイトリクエストフォージェリーに対する完全な防御策ではないことを忘れないでください。

#### ★ method\_missing を応用したメソッドの注意点

動的な言語である Ruby において、respond\_to? の様なメソッドは必須です。何度この respond\_to? メソッドや、is\_a? メソッドを使って、扱っているオブジェクトが目的のメソッドを提供しているか、目的のクラスであるかを確認したことでしょう。

しかし、多くの方が大切なことを忘れています。次の method\_missing メソッドを利用したクラスの例を見てみましょう。

```
-----  
class Dog  
  def method_missing(method, *args, &block)  
    if method.to_s =~ /^bark/  
      puts "wanwan!"  
    else  
      super  
    end  
  end  
end  
  
pochi = Dog.new  
pochi.bark      # => "wanwan!"  
pochi.bark!    # => "wanwan!"  
pochi.bark_and_run # => "wanwan!"  
-----
```

method\_missing メソッドについては知っていますよね。上の例では、Dog クラスのインスタンス pochi を生成して、bark、bark! と、bark\_and\_run という存在しないメソッドを呼び出しています。呼び出されると、method\_missing メソッドが呼び出され、正規表現の条件から、"wanwan!" という String が表示されます。

しかし、以下の様に、method\_missing メソッドを応用したメソッドに対して respond\_to? を使うとどうでしょうか。

```
-----  
pochi.respond_to? :bark # => false  
pochi.bark # => "wanwan!"  
-----
```

respond\_to? メソッドは、bark メソッドが存在しないことを意味する false を返してしまいます。この respond\_to? メソッドの戻り値を適切にしなければなりません。私の場合は、このクラスを次の様に変更しました。

```
-----  
class Dog  
  METHOD_BARK = /^bark/  
  
  def respond_to?(method)  
    return true if method.to_s =~ METHOD_BARK  
    super  
  end  
  
  def method_missing(method, *args, &block)  
    if method.to_s =~ METHOD_BARK  
      puts "wanwan!"  
    else  
      super  
    end  
  end  
end  
  
pochi = Dog.new  
pochi.respond_to?(:bark) # => true  
pochi.bark # => "wanwan!"  
-----
```

これでいいでしょう。この問題は Rails 自体を含め、多くのプログラムで散見されます。例えば、respond\_to? メソッドで、find\_by\_name などのメソッドの存在を確認してみるとわかります。

Ruby は非常に柔軟な素晴らしい言語ですが、気をつけなければこういった問題を招きかねません。

ちなみに Rails 2.1 では、この問題が解決され、respond\_to? メソッドで find\_by\_\* メソッドの存在を確認できるようになりました。

#### ★ PostgreSQL 7.4 以上に対応

Rails 2.0 までは、PostgreSQL アダプターはバージョン 8.1 から 8.3 にしか対応していませんでしたが、Rails 2.1 では、バージョン 7.4 から 8.3 に対応しています。

# 第14章

## 変更履歴 (“CHANGELOG”)

### ★ ActionMailer

- return-path ヘッダーが無視される問題 #7572 を修正 [joost]
- より冗長でないメールログ : :info は受信者のみ、全メール内容は :debug のみ。 #8000 [iaddict, Tarmo Tänav]
- TMail を 1.2.1 にバージョンアップ [raasdnil]
- ActionMailer::TestCase#setup で super を呼び出さなくても良くなりました。 [jamesgolick]

### ★ ActionPack

- InstanceTag#default\_time\_from\_options が DateTime にオーバーフローする [Geoff Buesing]
- セッション追跡なしでフォージェリー対策が使える様に修正 #139 [Peter Jones]
- session(:on) で、セッション管理が無効にされたコントローラーのサブクラスでセッション管理を有効にできる機能を追加 #136 [Peter Jones]
- リクエストフォージェリー対策で request.format から Content-Type を使う様に変更し、“#{request.uri}.xml” に POST してもバイパスされなくなりました。 [rick]
- InstanceTag#default\_time\_from\_options のハッシュで Time.current をデフォルトにし、システムが春のサマータイム時はハッシュの設定を優先 [Geoff Buesing]
- config.time\_zone が指定されている場合の select\_date のデフォルト値を Time.zone.today に設定 [Geoff Buesing]
- 生 HTML が TextHelper#text\_field の値として使われると壊れるのを修正 #80 [mchenryc, Kevin Glowacz]
- テスト中でアクションが rescue\_action パスを使うか、インラインで例外を出すかコントロールする ActionController::TestCase#rescue\_action\_in\_public! を追加（テストに便利） [DHH]
- コントローラーからビューにコピーされるインスタンス変数を減らしました。 [Pratik]
- config.time\_zone が設定されている場合の select\_datetime と select\_time のデフォルト値を Time.zone.now に設定 [Geoff Buesing]

- ActionController::Base#view\_controller\_internals フラグを削除 [Pratik]
- caches\_page メソッドに条件オプションを追加 [Paul Horsfall]
- ActionView に欠けていたテンプレートロジックを移動 [Pratik]
- 新規に ActionView::InlineTemplate 追加 [Pratik]
- Mime::JSON のリクエストでポストされた JSON の内容を自動的にパース [rick]

```
-----  
POST /posts  
  { "post": { "title": "Breaking News" } }  
  
def create  
  @post = Post.create params[:post]  
  # ...  
end  
-----
```

- ERB に、HTML 中に JSON 文字列を埋め込む際にエスケープする json\_escape メソッドを追加。 [rick]
-